Blue Gene/P Universal Performance Counters

Bob Walkup (walkup@us.ibm.com)

256 counters, 64 bits each; hardware unit on the BG/P chip

72 counters are in the clock-x1 domain (ppc450 core: fpu, fp load/store, …)
184 counters are in the clock-x2 domain (L2, L3, memory, networks)

counters in the clock-x1 domain are specific to each core
counters in the clock-x2 domain are mostly shared across the node

The counter mode and trigger method are programmable:

mode 0 : info on cores 0 and 1 for the clock-x1 counters
plus a set of 184 counters in the clock-x2 domain

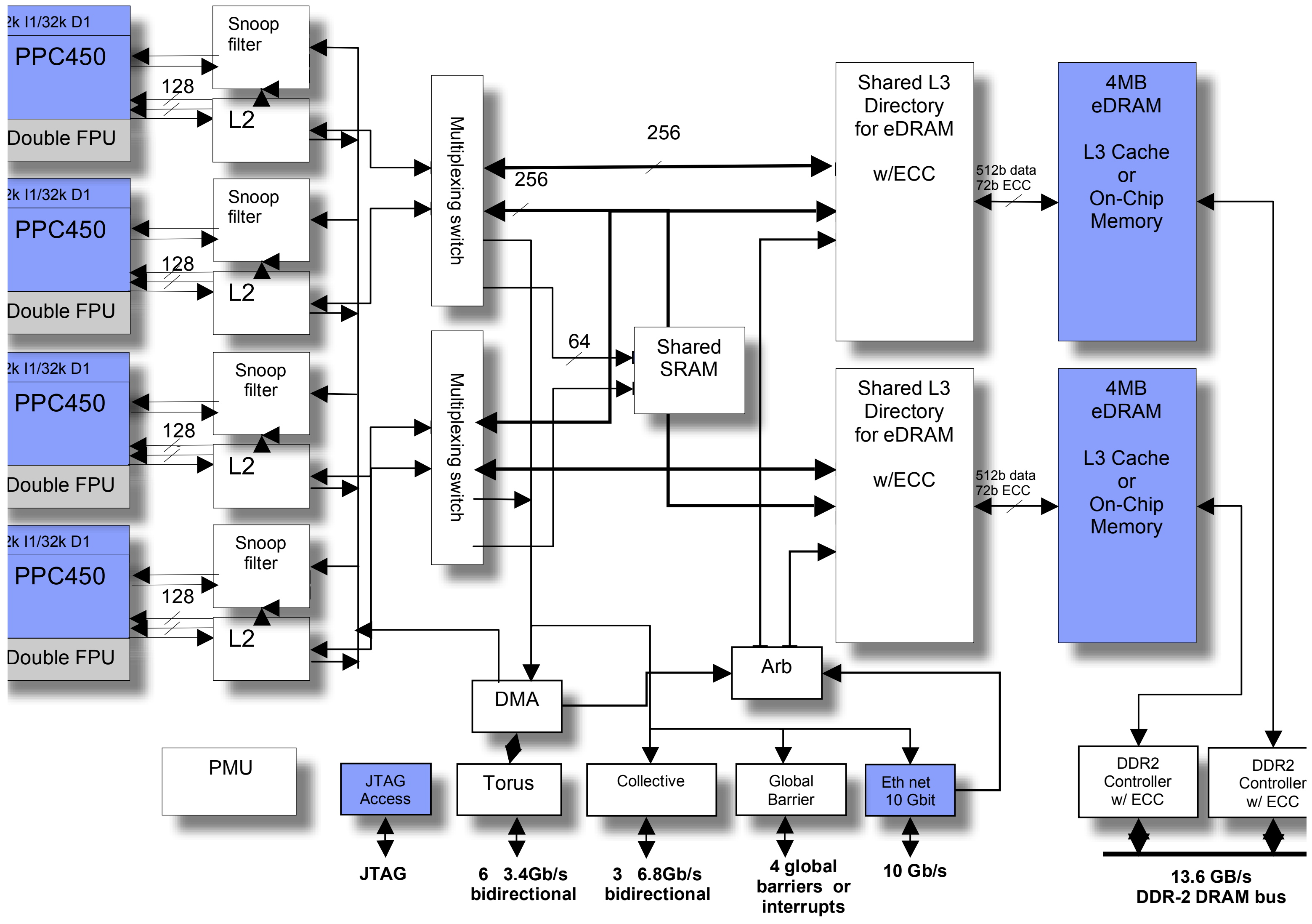
mode 1 : info on cores 2 and 3 for the clock-x1 counters
plus a different set of 184 counters in the clock-x2 domain

modes 2 and 3 : primarily intended for hardware designers

trigger methods: rising edge, default edge, falling edge, level high, level low

BGP counters are tied to hardware resources, either specific to a core or shared across the node. There is no process or thread-level context, but processes and threads are pinned to specific cores.

Question: What is “universal” about the BGP counters?



2k I1/32k D1
PPC450
Double FPU
Snoop filter
L2
128
Multiplexing switch
256
256
64
Shared SRAM
Shared L3 Directory for eDRAM w/ECC
512b data 72b ECC
4MB eDRAM L3 Cache or On-Chip Memory
Arb
DMA
PMU
JTAG Access
Torus
Collective
Global Barrier
Eth net 10 Gbit
DDR2 Controller w/ ECC
JTAG
6 3.4Gb/s bidirectional
3 6.8Gb/s bidirectional
4 global barriers or interrupts
10 Gb/s
13.6 GB/s DDR-2 DRAM bus

How to Access the Counters

The BGP_UPC interface definitions and list of events are in:

/bgsys/drivers/ppcfloor/arch/include/spi/UPC.h
/bgsys/drivers/ppcfloor/arch/include/spi/UPC_Events.h

Example:

```
// every process on the node calls BGP_UPC_Initialize()
BGP_UPC_Initialize();

// just one rank per node sets the counter config and zeros the counters
if (local_rank == 0) {
  BGP_UPC_Initialize_Counter_Config(counter_mode, counter_trigger);
  BGP_UPC_Zero_Counter_Values();
  BGP_UPC_Start(0);
}

MPI_Barrier(local_comm);  // communicator local to the node

do work …

MPI_Barrier(local_comm);

if (local_rank == 0) {
  BGP_UPC_Stop();
  BGP_UPC_Read_Counter_Values(&counter_data,
                              sizeof(struct CounterStruct),
                              BGP_UPC_READ_EXCLUSIVE);
  Save the counter values from the counter_data structure …
  BGP_UPC_Start(0);
}
```

counter_mode = 0, 1, 2, 3 (plus some others … see UPC.h)
counter_trigger = BGP_UPC_CFG_LEVEL_HIGH, BGP_UPC_CFG_EDGE_DEFAULT

BGP UPC Counter Data Structure

```
struct CounterStruct {
  int32_t rank;                          // Rank
  int32_t core;                          // Core
  int32_t upc_number;                    // UPC Number
  int32_t number_processes_per_upc;   // Number of processes per UPC unit
  BGP_UPC_Mode_t mode;                   // User mode
  int32_t number_of_counters;         // Number of counter values returned
  char location[24];                     // Location
  int64_t elapsed_time;                  // Elapsed time
  uint32_t reserved_1;                   // Reserved for alignment
  uint32_t reserved_2;                   // Reserved for alignment
  int64_t values[256];                   // Counter values
} counter_data;
```

Basic operation is BGP_UPC_Read_Counter_Values(&counter_data, …). It fills out a structure including 256 counter values, 64 bits each.

Caveats:

(1) Reading all of the counters takes a long time … of order 10**4 cycles. So in practice, you can only use the counters for coarse-grained measurements.

(2) The BGP headers (UPC.h) require the GNU compiler (mpicc, powerpc-bgp-linux-gcc) for compilation. So it is best to wrap the counter routines in separately compiled source.

What to Count and How to Count It

For starters I recommend modes 0 and 1, using the level high trigger or the default edge trigger. Normally you get all 256 counter values.

Some of the "useful" clock-x1 counters for mode 0 are:

counter#	label	
17	BGP_PU0_DATA_LOADS	
18	BGP_PU0_DATA_STORES	
22	BGP_PU0_FPU_ADD_SUB_1	
23	BGP_PU0_FPU_MULT_1	
24	BGP_PU0_FPU_FMA_2	
25	BGP_PU0_FPU_DIV_1	
26	BGP_PU0_FPU_OTHER_NON_STORAGE_OPS	
27	BGP_PU0_FPU_ADD_SUB_2	(SIMD)
28	BGP_PU0_FPU_MULT_2	(SIMD)
29	BGP_PU0_FPU_FMA_4	(SIMD)
30	BGP_PU0_FPU_DUAL_PIPE_OTHER_NON_STORAGE_OPS	(SIMD)
31	BGP_PU0_FPU_QUADWORD_LOADS	(SIMD)
32	BGP_PU0_FPU_OTHER_LOADS	
33	BGP_PU0_FPU_QUADWORD_STORES	(SIMD)
34	BGP_PU0_FPU_OTHER_STORES	

The counters above have PU0 in the name (processor unit 0 = core 0). The analogous counters for core 1 are counters 52,53,57-69. You can use these counters to construct a weighted "floating-point operation" count for cores 0 and 1. With counter-mode 1, the same counts are accumulated for cores 2 and 3. It takes both modes (normally two separate jobs) to get counts from all four cores.

Some counters in the clock-x1 domain that are not very "useful" are the dcache miss and hit counters (15,16 for core 0 or 2; 50, 51 for core 1 or 3). Some reasons: stores for data not in L1 count as a dcache miss; for data streaming from memory all loads are dcache misses because the data was not in L1 dcache when the load was issued. There is no clear relationship between dcache misses and application performance.

Some Counters from the Clock-X2 Domain

Counters in the clock-x2 domain are shared across the node: and at the L3 level you have to add up contributions from the two sections. Mode 0 has the most "useful" counters for the memory subsystem.

Examples:

Number of 128-byte cache lines loaded from DDR = counters 155 + 175:

mode	counter#	label
0	155	BGP_L3_M0_MH_DDR_FETCHES
0	175	BGP_L3_M1_MH_DDR_FETCHES

Number of 128-byte cache lines stored to DDR = counters 154 + 174:

mode	counter#	label
0	154	BGP_L3_M0_MH_DDR_STORES
0	174	BGP_L3_M1_MH_DDR_STORES

Note: some counters count events, and other counters count "cycles", but one cycle in the clock-x2 domain = two processor cycles. Example: counter 80 (mode 0) "BGP_PU0_L2_CYCLES_READ_REQUEST_PENDING" … with trigger = level high, is the number of memory-bus cycles where the L2 unit attached to core 0 is waiting on a read request. You have to multiply by two to get processor cycles.

Some of the interesting counters provide data on prefetching, L2 read requests, L2-L3 read requests, and DDR loads/stores. Memory events at L2, L3 and memory involve 128-byte cache lines. Any process or thread running on the node can (will) trigger the shared counters in the clock-x2 domain, and this needs to be remembered to properly interpret the data.

Torus Network Counters

Counter mode 0 includes a number of "useful" torus network counters:

mode	counter#	label
0	194	BGP_TORUS_XP_PACKETS
0	195	BGP_TORUS_XP_32BCHUNKS

etc.

XP is for the +X direction; XM is for the –X direction, and there are similar counters for Y and Z.

One packet = 256 bytes, including header data. The packet counter increments once for every packet sent (but not received) by the torus hardware on the node. The "32-byte chunk" counters increment once for every 32-byte chunk sent by the torus hardware on the node. For very large messages: #packets = 8*#32Bchunks. For very small messages #packets = 2*#32Bchunks. Packets or chunks that are received by the torus hardware are not counted at all.

Counter mode 0 also includes DMA event counters (#206-215).

Counter mode 2 includes additional torus counters, including ones that are triggered if there are no "tokens": mode 2, counter 134, label = BGP_TORUS_XP_NO_TOKENS (and similar counters for -X, +/-Y, +/-Z). These counters indicate contention for torus resources.

Collective Network Counters

Counter modes 0 and 1 provide some insight into utilization of the collective network. For mode 0, you can look at:

mode	counter#	label
0	248	BGP_COL_INJECT_VC0_HEADER
0	249	BGP_COL_INJECT_VC1_HEADER
0	250	BGP_COL_RECEPTION_VC0_PACKET_ADDED
0	251	BGP_COL_RECEPTION_VC1_PACKET_ADDED

There are two virtual channels: VC0 is for the user's application including things like MPI_Allreduce; VC1 is for use by the kernel (I/O activity). For the collective network, headers and data-packets are counted separately. There is normally one header per data-packet, and data-packets contain up to 256 bytes. So counter 248 provides an indicator of data sent over the collective network in the user's space, and counter 249 indicates data shipped over the collective network by the kernel (i.e. data written to the file-system). Similarly counter 250 indicates the number of incoming packets for virtual-channel 0, the user's application space, while counter 251 indicates packets received on the network by the kernel (i.e. data read from the file-system). Normally packets "added" to the reception queue are actually received, but if you want to make absolutely sure, you can use counter mode 1, which offers additional collective network counters:

mode	counter#	label
1	124	BGP_COL_INJECT_VC0_PACKET_TAKEN
1	125	BGP_COL_INJECT_VC1_PACKET_TAKEN
1	128	BGP_COL_RECEPTION_VC0_PAYLOAD_TAKEN
1	129	BGP_COL_RECEPTION_VC1_PAYLOAD_TAKEN

For virtual-channel 1, each data-packet contains some metadata for the file-system: there are about 240 bytes of real data per "packet" on the collective network. One can use these counters (virtual channel 1) to provide bytes read and bytes written in file I/O operations.

How to Compute Flops?

The floating-point operation counts are tied to each core. Normal addition, subtraction, and multiplication count as one op each. A normal fmadd operation counts as two ops. SIMD addition, subtraction, and multiplication count as two ops. SIMD fmadd operations count as four ops. What about division? A normal fdiv instruction takes about 30 cycles (for doubles), but the compiler can generate pipeline-able normal or SIMD instructions to do an array of division operations, using the floating-point reciprocal estimate instruction, and Newton's method to refine the estimate. My suggestion: choose a weighting factor for fdiv such that the "operation count" is equal for the two methods. For double-precision real numbers, this would assign fdiv a weighting factor of 13. BG/P does not have a hardware sqrt() instruction, but how to handle fdiv is an open, debatable question. Also, I count the instructions lumped under the label "fpu_other_non_storage_ops". Those include things like fsel() [used in floating-point min()/max()], fctiwz [floating-point convert to integer], fre [floating-point reciprocal estimate], and the corresponding SIMD instructions.

Example:

```
fp_op_count =  1LL * counter_data.values[22]   // 1st core fadd, fsub
          +  1LL * counter_data.values[23]   // 1st core fmul
          +  2LL * counter_data.values[24]   // 1st core fmadd family
          + 13LL * counter_data.values[25]   // 1st core fdiv
          +  1LL * counter_data.values[26]   // 1st core other fp non-storage ops
          +  2LL * counter_data.values[27]   // 1st core SIMD fpadd, fpsub
          +  2LL * counter_data.values[28]   // 1st core SIMD fpmul
          +  4LL * counter_data.values[29]   // 1st core SIMD fpmadd family
          +  2LL * counter_data.values[30]   // 1st core SIMD fp non-storage ops
```

Other Counter Interfaces for BGP

All hardware counter interfaces for BG/P are layered on top of BGP_UPC.

The BGP_UPC layer is provided, so you can write your own interfaces.

HPC Toolkit provides documentation in HPM_ug.pdf ; there is no hpmcount or hpmstat for BG/P, just libhpm.a. The env variable HPM_EVENT_SET is used to set the counter mode 0, 1, 2, 3; default value is 0. The default trigger method was previously "edge rise" (should be changed and made user settable). Currently (7/22/2008) being updated to fix issues.

```
#include <libhpm.h>
…
hpmInit(rank, program);
hpmTstart(number, label);
do_work();
hpmTstop(number);
hpmTerminate(rank);      // prints counter values etc.
```

PAPI 3.0.9 has been ported to BG/P. Some information has been posted by Argonne National Labs:

http://trac.mcs.anl.gov/projects/performance/wiki/Machines

I recommend using the BGP_UPC layer directly due to the numerous machine-specific properties of the BG/P counters.

Some Experimental Tools

libmpihpm.a Uses the MPI profiling interface, starts the BGP UPC counters in MPI_Init(), stops them in MPI_Finalize(), and produces two counter ouput files:

(1) One text summary with min, max, avg counter values

(2) One binary file with all counter data from every node

There is a little utility (getcounts) that can be used to pull out the data for a given node from the aggregate binary file. Nodes are numbered in x, y, z order on the partition that the job ran on.

This utility provides aggregate Flops for the whole job, from start to finish, along with MPI statistics; but can't be used to measure specific code blocks.

Primative libhpm.a – A simple start/stop interface that can be called from Fortran, C, C++ to get counts around specific code blocks, with one output file per node. Set env variables:

BGP_COUNTER_MODE=0,1,2,3 (default = 0)
BGP_COUNTER_TRIGGER={edge, high} (default method = high)

Fortran interface:

```
call hpm_init()            ! one time to initialize counters
call hpm_start('label')    ! start counting a labeled block
call hpm_stop ('label')    ! stop  counting a labeled block
call hpm_print()           ! print counter values and labels once at the end
```

C interface (add extern "C" for C++):

```
void HPM_Init(void);
void HPM_Start(char * label);
void HPM_Stop(char * label);
void HPM_Print(void);
```